AuthService

September 4, 2026

AuthService

Kind: Service

Source: `atloria-monorepo/apps/api/src/auth/auth.service.ts`

`AuthService` encapsulates the API's authentication lifecycle, including registration, email verification, password recovery, credential login, token refresh, and logout. It coordinates user validation and user-profile retrieval while returning DTOs suitable for authentication controllers and API consumers. In the NestJS backend, it serves as the central business-logic layer between auth endpoints, user persistence, and token/email integrations.

Methods

Method	Signature	Returns	Description
`register`	`register(registerDto: RegisterDto)`	`Promise<AuthResponseDto>`	Register a new user
`verifyEmail`	`verifyEmail(token: string)`	`Promise<{ verified: boolean; email: string }>`	Verify a user's email address using the emailed token.
`resendVerification`	`resendVerification(userId: string)`	`Promise<{ sent: boolean; alreadyVerified?: boolean }>`	Re-send the verification email for a logged-in, unverified user.
`forgotPassword`	`forgotPassword(email: string)`	`Promise<{ message: string }>`	Start a password reset.
`resetPassword`	`resetPassword(token: string, newPassword: string)`	`Promise<{ success: boolean }>`	Complete a password reset with a valid,

Method	Signature	Returns	Description
			unexpired token.
`login`	`login(loginDto: LoginDto)`	`Promise<AuthResponseDto>`	Login user
`refresh`	`refresh(refreshToken: string)`	`Promise<AuthResponseDto>`	Refresh access token
`logout`	`logout(userId: string)`	`Promise<void>`	Logout user
`getMe`	`getMe(userId: string)`	`Promise<UserResponseDto>`	Get current user
`validateUser`	`validateUser(email: string, password: string)`	`Promise<User	null>`	
`issueTokensForUser`	`issueTokensForUser(user: User)`	`Promise<AuthResponseDto>`	A1 SSO: mint the SAME access+refresh token pair as a password login for an already-authenticated user (e.g.
`revokeAllForUser`	`revokeAllForUser(userId: string)`	`Promise<void>`	A2 deprovisioning: revoke every active session for a user by clearing their refresh token in Redis.

Dependencies

- `PrismaService`
- `RedisService`
- `JwtService`
- `ConfigService`
- `EmailService`
- `AuditService` *(optional)*

Where it refuses work

- `AuthService` stops the work with `UnauthorizedException` when `!user` — “User not found”, in 2 places.
- `AuthService` stops the work with `ConflictException` when `existingUser` — “User with this email already exists”.
- `AuthService` stops the work with `BadRequestException` when `!token` — “Verification token is required”.
- `AuthService` stops the work with `BadRequestException` when `!user` — “Invalid or already-used verification token”.
- `AuthService` stops the work with `BadRequestException` when `!user` — “User not found”.
- `AuthService` stops the work with `BadRequestException` when `!token` — “Reset token is required”.

When something fails

- `AuthService` handles failure in 2 places: it logs it and continues in 1, and lets it reach the caller in 1.

Diagram

```
sequenceDiagram
    participant Client
    participant AuthController
    participant AuthService
    participant UsersService
    participant TokenService
    participant EmailService

    Client->>AuthController: register(credentials)
    AuthController->>AuthService: register()
    AuthService->>UsersService: create user
    AuthService->>TokenService: generate auth tokens
    AuthService->>EmailService: send verification email
    AuthService-->>AuthController: AuthResponseDto
    AuthController-->>Client: authentication response

    Client->>AuthController: login(credentials)
    AuthController->>AuthService: login()
    AuthService->>AuthService: validateUser()
    AuthService->>TokenService: generate auth tokens
    AuthService-->>Client: AuthResponseDto

    Client->>AuthController: refresh(refreshToken)
    AuthController->>AuthService: refresh()
```

```
    AuthService->>TokenService: validate and rotate token
    AuthService-->>Client: AuthResponseDto
```

Usage

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { AuthService } from './auth.service';

@Injectable()
export class SessionFacade {
  constructor(private readonly authService: AuthService) {}

  async signIn(email: string, password: string) {
    const user = await this.authService.validateUser(email, password);

    if (!user) {
      throw new UnauthorizedException('Invalid email or password');
    }

    return this.authService.login(user);
  }

  async registerAccount(registerDto: {
    email: string;
    password: string;
    firstName: string;
    lastName: string;
  }) {
    return this.authService.register(registerDto);
  }

  async refreshSession(refreshToken: string) {
    return this.authService.refresh(refreshToken);
  }

  async requestPasswordReset(email: string) {
    return this.authService.forgotPassword(email);
  }
}
```

AI Coding Instructions

- Keep authentication business logic in `AuthService`; controllers should only validate request DTOs, extract request context, and delegate to service methods.
- Use `validateUser()` before issuing credentials through `login()`; never expose whether a password specifically failed in public error responses.

- Preserve the `AuthResponseDto` and `UserResponseDto` response contracts when changing token payloads or user fields.
- Ensure email verification, password-reset, and refresh-token flows validate token expiry and ownership before modifying user state.
- When adding auth providers or token strategies, integrate them through the existing user lookup, token generation, and logout/refresh lifecycle rather than duplicating session logic.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `RedisService`
- DEPENDS_ON → `jwtservice`
- DEPENDS_ON → `configservice`
- DEPENDS_ON → `EmailService`
- DEPENDS_ON → `AuditService`

Referenced By

- `AuthController` (DEPENDS_ON)
- `AuthModule` (MODULE_PROVIDES)
- `AuthModule` (MODULE_EXPORTS)
- `SamlController` (DEPENDS_ON)
- `ScimService` (DEPENDS_ON)